1. Expresiones Lambda:

Son funciones anónimas, esenciales en la programación declarativa, su sintaxis es: **(parámetros) -> expresión**

1. Referencias a métodos (Interfaces Aumentadas):

Las interfaces en java 8 pueden declarar métodos con implementación gracias a dos mejoras

* 1. default methods: Permite definir métodos default en la interfaz, de manera que si una clase implementa una interfaz que tiene métodos default y estos no se sobre-escriben es como que si la clase estuviera heredando ese método (herencia múltiple), realmente no se hereda todo un objeto más bien solo ciertos comportamientos default de un objeto en cuestión, El modificador default solamente puede ser utilizada en interfaces. Estas nuevas “libertades” le dan un nuevo propósito a las interfaces, y hasta cierto punto previenen que se tenga código repetido a lo largo de varias aplicaciones. Ejemplo:

public static <T> Stream<T> of(T... values) {

return Arrays.stream(values);

}

* 1. Un nuevo operador “::“ el dos puntos doble permite hacer referencia a un método de una clase que necesitemos y poder pasarlo como parámetro de una función.
  2. Static methods: Deja poner métodos estáticos ya implementados en las interfaces de forma que podríamos poner métodos de utilidades en las interfaces sin repetir código en las clases que lo implementan, siempre con la posibilidad de poner la anotación @Override y re-escribir estos métodos

public class App {

public static void main(String[] args) {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* java < 8 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

List<String> letras = Arrays.asList("C", "a", "A", "b");

Collections.sort(letras, new Comparator<String>() {

@Override

public int compare(String s1, String s2) {

return s1.compareToIgnoreCase(s2);

}

});

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* java >= 8 \*\*\*\*\*\*\*\*\*\*\*\*\*

Collections.sort(letras, String::compareToIgnoreCase);

System.out.println(letras);

}

}

1. Streams: Una forma óptima de trabajar con estructuras de datos
2. Nuevo API para fechas y tiempo

* Introducen clases como **Period** que sirve para representar un periodo de tiempo como “3 meses y 4 días”
* ZonedDateTime para representar un timestamp con zona horaria
* LocalDateTime de la siguiente forma para añadir tiempo a un timestamp

LocatedDateTime vacaciones = LocalDateTime.now()

.plusHours(24)

.plusMinutes(30);

1. Completable Future: La forma en como abstraemos la programación asíncrona, para hacer un comportamiento “race”
2. Optionals: Es una alternativa para evitar los famosos NullPointerException y sirve básicamente para representar cuando un valor está presente o vacío logrando cierta consistencia al verificar, ya que no necesariamente se tiene que comparar un objeto con null